Mult-GUP based Image Feature Matching

Liang Xu
Department of Electrical and Computer Engineering
Virginia Commonwealth University
Richmond, Virginia
Email: xul4@vcu.edu

***Abstract*—During the Ramhack this year (2016), elephant insurence proposed one of the channenges called sub-images. The goal of the challenge is to match a sub image to a big image in a image pool. The sub image is a small part of the big image. Example of the source image is Figure [1]**

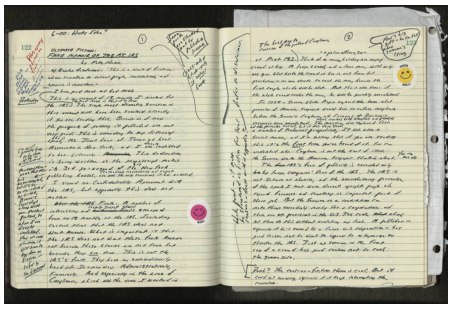

Fig. 1. Source Image

Example of the target image is Figure [2]:

Fig. 2. Target Image

During the challenge we use the eeathre matching algorithm with FLANN(Fast Approximate Nearest Neighbor Search Library) in OpenCV. OpenCV (Open Source Computer Vision Library: http://opencv.org) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. The final challenge is to match 75 pictures. Our algorithm took around 2 hours to finish the challenge which takes too much time by using the serial computation (CPU).

So for this final project, I am thinking to continue work on the challenge using more advanced computation method, for example using GPU to parallel the computation.

CUDA and open-CV will be used, and more detailed comparison will be proposed in the future investigation.

***Index Terms*—GPU, OPENCV, Picture matching**

THIS project is going to speed up the original program. The original program run about 5 hours under 400 min Hessian distance. The average running time is 18191416089 micro seconds. Let treat this as benchmark. I will propose 4 different method to speed up the program. 1. Better Programming. 2, Use Multi Thread. 3, Single GUP. 4, Multi-GUP. In the following sections, I will discuss about each of them. The original target and source each have all the 75 pictures.

I. Running Algorithm

This program is using the OpenCV library.

Keypoints

Using the SurfFeatureDetector to extract all the keypoints. Figure [3], Figure [4]

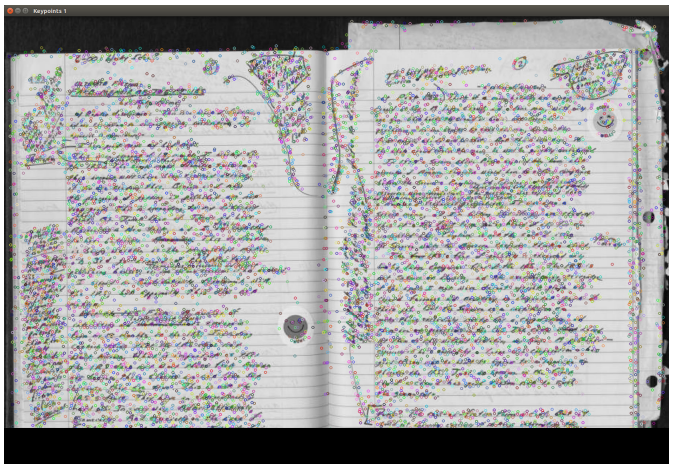

Fig. 3. Source Keypoints

Fig. 4. Target Keypoints

Hessian Value

Different Hessian value will generate different number of keypoints. Following is the three source image with different Hessian value.Figure [5], Figure [6],Figure [7]

Different Hessian value means different complexity. This project is using 400 for Hessian value.

A. descriptors and Matching method

Figure [8],Figure [9]

II. Original Program

The original program has 3 expensive function from opencv.
1.img = imread();
2.detector.detect( );
3.extractor.compute( );
4.matcher.match( );
The original program running:

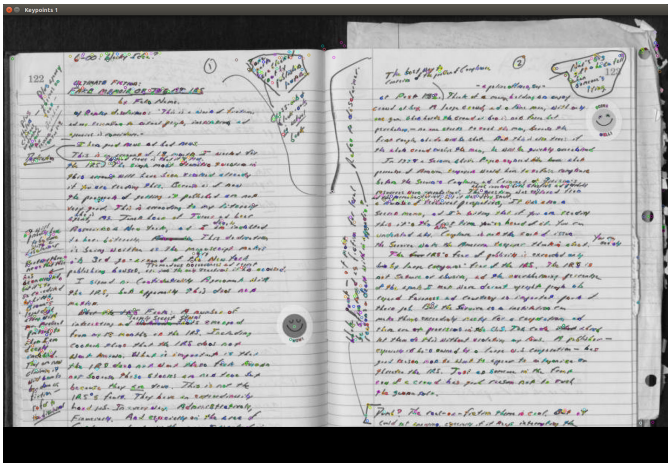

Fig. 5. Hessian = 4000

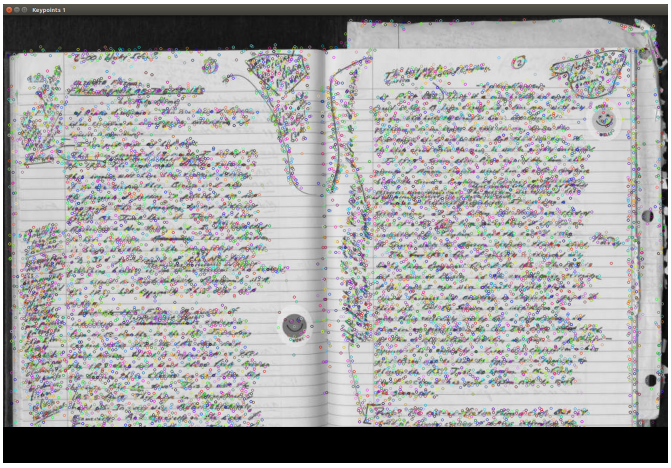

Fig. 6. Hessian = 400

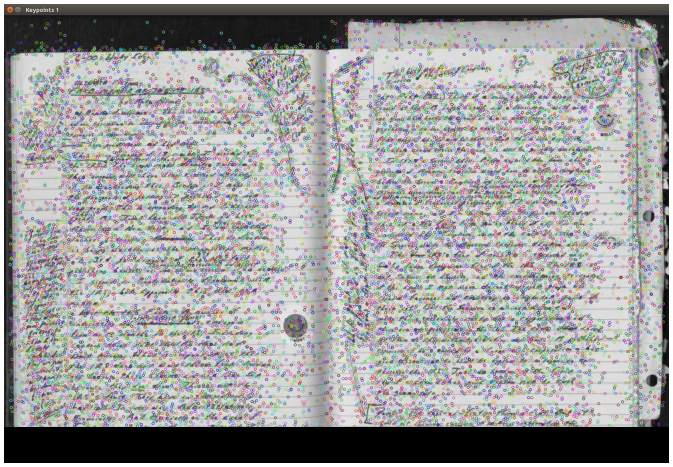

Fig. 7. Hessian = 20

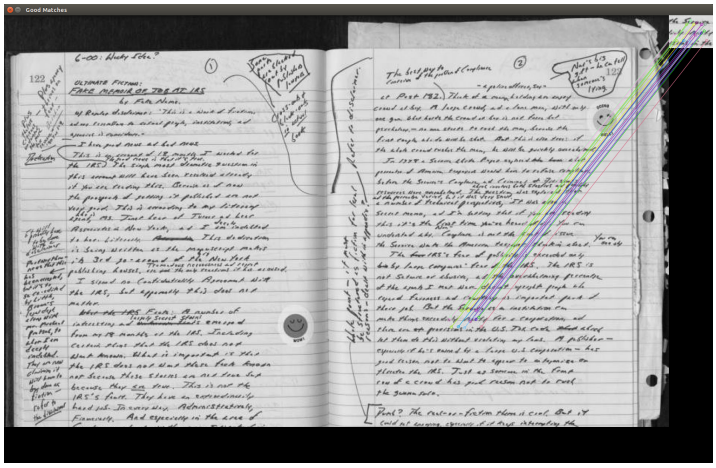

Fig. 8. Match Keypoints

for(Source Image)
— img1 = imread();
— for (Target Image)
———img2 = imread();
———detector.detect( );
———extractor.compute( );
———matcher.match( );
—-end
end

for 75 image the loop will run 75 * 75 times, and all the imread nd derector and extractor are wasted. So the improved programming version is going to solve this issue.

III. Improved Version

The improved version will save all the image data in the memory.

for(Source Image)
— imgSource = imread();
—-detector.detect(imgSource);
—-extractor.compute(imgSource);
end
for(Target Image)
— imgTarget = imread();
—-detector.detect(imgTarget);
—-extractor.compute(imgTarget);
end
for (Source Image In Memory)
—-for(Target Image in Memory)
———matcher.match( );
end

The complexity of this program is the same as the original one. But this improved version save the computation time to recalcuate the detector and extractor. speed up is 13.8312. The average time of this improved version is 1315246048 micro second which is 21.92 minutes.

IV. Multi Thread Version

The multi Thread version of the program is based on the improved version.

Thread
for(Source Image)
— imgSource = imread();
—-detector.detect(imgSource);
—-extractor.compute(imgSource);

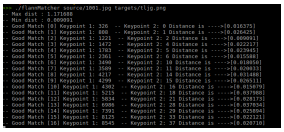

Fig. 9. Keypoint Distance

end
for(Target Image)
— imgTarget = imread();
—-detector.detect(imgTarget);
—-extractor.compute(imgTarget);
end
for (Source Image In Memory)
—-for(Target Image in Memory)
———matcher.match( );
end
//end thread this is the flow chart Figure [10]

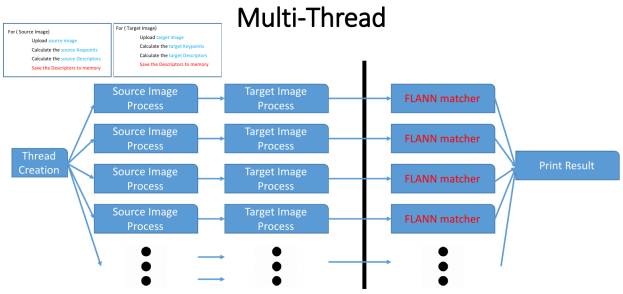

Fig. 10. Multi Thread

For each thread is rung the multi thread version of the original program. Speedup is 132.444 this version takes about 137352134 micro sceconds, it is around 2.29 minutes.

V. GPU Version

The multi Thread version of the program is based on the improved version.
for(Source Image)
— imgSource = imread(); ———GPU Version
—-detector.detect(imgSource); —GPU Version
—-extractor.compute(imgSource);– GPU Version
end
for(Target Image)
— imgTarget = imread(); ——— -GUP Version
—-detector.detect(imgTarget); —GUP Version
—-extractor.compute(imgTarget);–GUP Version
end
for (Source Image In Memory)
—-for(Target Image in Memory)
———matcher.match( ); ———GUP Version
end this is the flow chart Figure [11]
Speedup is 645.581 this version takes about 28178377 micro sceconds, it is around 28 seconds.
thread(i)
Call Different GUP for this entire thread
for(Source Image)

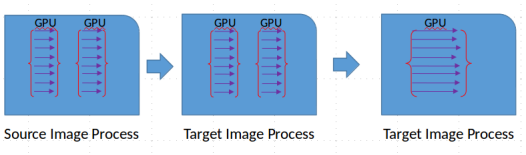

Fig. 11. GPU Version

— imgSource = imread(); ———GPU Version
—-detector.detect(imgSource); —GPU Version
—-extractor.compute(imgSource);– GPU Version
end
for(Target Image)
— imgTarget = imread(); ——— -GUP Version
—-detector.detect(imgTarget); —GUP Version
—-extractor.compute(imgTarget);–GUP Version
end
for (Source Image In Memory)
—-for(Target Image in Memory)
———matcher.match( ); ———GUP Version
end
// end thread this is the flow chart Figure [12]

VI. Multi GPU Version

The multi Thread version of the program is based on the improved version.

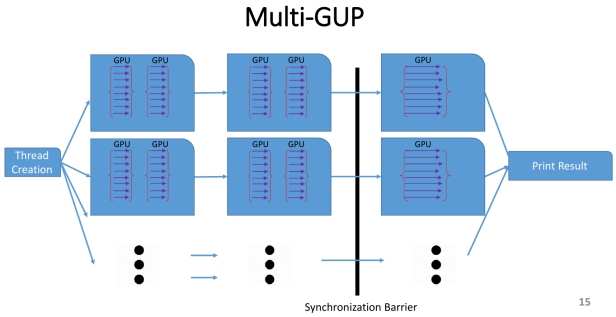

Fig. 12. Multi GPU

Speedup is 1103.013 this version takes about 16492464 micro sceconds, it is around 16 seconds.

VII. Result

This part is the result of this project Figure [13]

Mean(micro S)	Std	Speedup
18333352721.7 (5.1 hours)	95949909	1
1327316537.5 (22.1 minutes)	12260675	13.81
144935912.6 * (2.4 minutes)	7677366	126.49
28056436 ** (28 seconds)	797203	653.44
16347417.5*** (16 seconds)	415412	1121.48

Fig. 13. Keypoint Distance

VIII. Conclusion

Two points I need to mention in the conclusion. One is try to balance the work load between the threads. Another is the memory and cache hierarchy.